### 8.1.4. A tranzakciók alaptevékenységei

Vizsgáljuk meg részletesen a tranzakció és adatbázis kölcsönhatását. A kölcsönhatásoknak három fontos színhelye van:

1. Az adatbázis elemeit tartalmazó lemezblokkok területe.

2. A pufferkezelő által használt virtuális vagy valós memóriaterület.

3. A tranzakció memóriaterülete.

Ahhoz, hogy a tranzakció egy adatbáziselemet beolvashasson, azt előbb memóriapuffer(ek)be kell behozni, ha még nincs ott. Ezt követően tudja a puffer(ek) tartalmát a tranzakció saját memóriaterületére beolvasni. Az adatbáziselem új értékének kiírása fordított sorrendben történik. Az új értéket a tranzakció alakítja ki saját memóriaterületén, majd ez az új érték másolódik át a megfelelő puffer(ek)be.

A pufferek tartalmát vagy azonnal lemezre lehet írni, vagy nem; az erre vonatkozó döntés általában a pufferkezelő joga. Amint már korábban láthattuk, a naplózó rendszer használatának egyik legfőbb lépése a rendszerhibákból való helyreállíthatóság biztosítása érdekében a pufferkezelő ösztönzése a pufferbeli blokkok megfelelő időpontban történő lemezre írására. Ugyanakkor a lemez I/O-műveletek számának csökkentésére az adatbázisrendszerek megengedik/megengedhetik a módosításoknak csak az illékony memóriában történő végrehajtását, legalábbis bizonyos ideig és arra megfelelő feltételek teljesülése esetén.

A naplózási algoritmusoknak és más tranzakció-kezelő algoritmusoknak részletes tanulmányozása során megfelelő jelölésekre lesz szükségünk, melyekkel a különböző területek közötti adatmozgást tudjuk leírni. A következő alapműveleteket fogjuk használni:

1. INPUT(X): Az *X* adatbáziselemet tartalmazó lemezblokk másolása a memória pufferbe.

2. READ(X,t): Az *X* adatbáziselem bemásolása a tranzakció *t* lokális változójába. Részletesebben, ha az *X* adatbáziselemet tartalmazó blokk nincs a memóriapufferben, akkor előbb végrehajtódik INPUT(X). Ezután kapja meg a *t* lokális változó az *X* értékét.

3. WRITE(X,t): A *t* lokális változó tartalma az *X* adatbáziselem memóriapufferbeli tartalmába másolódik. Részletesebben: ha az *X* adatbáziselemet tartalmazó blokk nincs a memóriapufferben, akkor előbb végrehajtódik INPUT(X). Ezután másolódik át a *t* lokális változó értéke a pufferbeli *X*-be.

4. OUTPUT(X): Az *X* adatbáziselemet tartalmazó puffer kimásolása lemezre.

A fenti műveleteknek addig van értelmük, amíg az adatbáziselemek elférnek egy-egy lemezblokkban és így egy-egy pufferben is. Ezt az esetet úgy is tekinthetjük, hogy az adatbáziselemek *pontosan* a blokkok. Adatbáziselem lehet az adatbázis egy-egy sora is. Mindaddig így tekinthetjük, amíg a relációséma nem engedi meg nagyobb („hosszabb”) sorok előfordulását, mint amennyi hely egy blokkban rendelkezésre áll. Ha az adatbáziselem több blokkot foglal el, akkor úgy is tekinthetjük, hogy az adatbáziselem minden blokkméretű része önmagában egy adatbáziselem. A naplózási mechanizmus, melyet arra használunk, hogy a tranzakció ne fejeződhessen be az *X* kiírása nélkül, atomos; azaz *X* összes blokkját vagy lemezre írja, vagy semmit sem ír ki. A továbbiakban a naplózási meggondolásokban úgy tekintjük, hogy:

• Az adatbáziselem nem nagyobb egy blokknál.

Fontos figyelembe venni, hogy ezen parancsokat kiadó komponensek különbözőek. A READ és WRITE utasításokat a tranzakciók használják, az INPUT és OUTPUT utasításokat a pufferkezelő alkalmazza, ezen túl, ahogy már láttuk, bizonyos feltételek esetén az OUTPUT utasítást a naplózási rendszer is használja.

**8.2. példa:** Annak bemutatására, hogy a tranzakció mikor és hogyan használja a fenti alapműveleteket, tegyük fel, hogy az adatbázis két, *A* és *B* eleme tartalmának, az adatbázis minden konzisztens állapotában meg kell egyeznie[[1]](#footnote-1).

A *T* tranzakció tartalmazza a következő két lépést:

A := A\*2;

B := B\*2;

Vegyük figyelembe, hogyha a tranzakcióra az egyetlen konzisztencia elvárás az
*A = B*, továbbá ha *T* korrekt adatbázis-állapotban indul, és tevékenységét rendszerhiba, valamint a párhuzamosan működő tranzakciókkal való kölcsönhatás nélkül be tudja fejezni, akkor az adatbázis befejezéskori állapotának is konzisztensnek kell lennie. Ekkor *T* megduplázva két azonos tartalmú elem értékét, kap két új, azonos értékű elemet.

*T* végrehajtása maga után vonja *A* és *B* lemezről való beolvasását, az aritmetikai műveletek a *T* lokális memória változóiban kerülnek végrehajtásra, végül *A* és *B* új értékei visszaírásra kerülnek a puffereikbe. *T*-t hat lényeges lépésből állónak tekinthetjük:

READ(A,t); t := t\*2; WRITE(A,t);

READ(B,t); t := t\*2; WRITE(B,t);

Ehhez még hozzáadódik az, hogy a pufferkezelő önállóan végrehajt OUTPUT lépéseket a pufferek tartalmának lemezre történő visszaírása végett. A 8.2. ábra a *T* elemi lépéseit és az őket követő, a pufferkezelő által végrehajtott OUTPUT utasításokat szemlélteti. Tegyük fel, hogy kezdetben *A = B =* 8. Az *A* és *B* pufferbeli és lemezen tárolt értékei és a *T* tranzakció *t* lokális változójának értékei lépésenként a következők:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Tevékenység | t | Mem A | Mem B | Lemez A | Lemez B |
| READ(A,t) | 18 | 18 |  | 18 | 18 |
| t := t\*2 | 16 | 18 |  | 18 | 18 |
| WRITE(A,t) | 16 | 16 |  | 18 | 18 |
| READ(B,t) | 18 | 16 | 18 | 18 | 18 |
| t := t\*2 | 16 | 16 | 18 | 18 | 18 |
| WRITE(B,t) | 16 | 16 | 16 | 18 | 18 |
| OUTPUT(A) | 16 | 16 | 16 | 16 | 18 |
| OUTPUT(B) | 16 | 16 | 16 | 16 | 16 |

**8.2. ábra.** A tranzakció lépései, és hatásuk a memóriában és a lemezen

*T* első lépésében beolvassa *A*-t, mely igény a pufferkezelőben kiváltja az INPUT(A) utasítást, ha *A* még nincs a pufferben. *A* értéke a READ utasítás hatására a *T* tranzakció memóriaterületére a *t* változóba is bemásolódik. A következő lépés megduplázza *t* tartalmát, ennek nincs hatása sem *A* pufferbeli, sem *A* lemezen tárolt értékére. A harmadik lépés írja *t*-t *A* pufferébe, s ennek nincs hatása *A* lemezen tárolt értékére. A következő három lépés ugyanez, csak *B*-re vonatkozóan. Végül az utolsó két lépésben másolódik *A* és *B* lemezre.

Figyeljük meg, hogy ezen lépések összességének végrehajtása alatt az adatbázis konzisztenciája megőrződik. Ha OUTPUT(A) végrehajtása előtt rendszerhiba fordul elő, akkor ennek nincs hatása a lemezen tárolt adatbázisra, az még olyan, mintha *T* egyáltalán nem is működött volna, s így a konzisztencia megőrződött. Ha rendszerhiba áll elő OUTPUT(A) végrehajtása után, de még OUTPUT(B) végrehajtása előtt, akkor az adatbázis inkonzisztens állapotban marad. Azt nem tudjuk megelőzni, hogy ilyen szituáció soha elő ne forduljon, de lépéseket tehetünk azért, hogy amikor mégis bekövetkezik, akkor a problémát elháríthassuk – vagy *A* és *B* értékének 8-ra való visszaállításával vagy mindkettő 16-ra növelésével. 

1. Természetesen feltehető a kérdés, hogy miért használnánk két különböző elemet, melyek tartalma mindig megegyezik ahelyett, hogy egyetlen elemet alkalmaznánk. Mindazonáltal, ennek az egyszerű numerikus megszorításnak a teljesítése jól jellemez nagyon sok valóságos megszorítást, mint például amikor előírják, hogy a repülőn az eladott helyek száma 10%-nál többel nem haladhatja meg a fedélzeten lévő ülések számát, vagy amikor előírják, hogy a bank kölcsönei összegének meg kell egyeznie a bank követelései összegével. [↑](#footnote-ref-1)